Chapter 1

Fully connected neural networks

1.1 Supervised learning in pills

A standard task in supervised machine learning can be summarized shortly as follows. Imagine we observe $x_1, \ldots, x_N$ feature vectors, living in $\mathbb{R}^D$ together with their labels $y_1, \ldots, y_N$. Our aim is to learn a function f_θ linking x_i to y_i, for all $i \in \{1, \ldots, N\}$ in such a way to make predictions $\hat{y}_i := f_\theta(x_i)$ being "as faithful as possible" to the observed labels $y_1, \ldots, y_N$. More formally y_i can be either a real/integer vector (regression) or a categorical variable in 1-to-K binary encoding (classification, K classes). Depending on the nature of y_i, one defines a loss function $L(y_i, \hat{y}_i)$ (e.g. mean squared error or cross-entropy) and seek to minimize the (empirical) expected loss with respect to θ

$$\min_\theta \left(\frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{y}_i) \right). \tag{1.1}$$

That is, we seek to estimate (or learn) a value of θ allowing us to minimize the average prediction error on the train data set.

Recall. *Although solving the above minimization problem is something that we need in order to "learn" a good predictor/classifier, that minimization only is an intermediate objective. The final objective is to be able to correctly predict/classify any new test data point x^*, via $f_\theta(x^*)$. In other terms we seek to avoid both* overfitting *and* underfitting.

In order to avoid overfitting, one might modify the above objective by adding some regularisation terms, such as ℓ_2 or ℓ_1 penalties, for instance, and/or monitor the loss on a validation data set in such a way to early stop the optimisation if needed (useful when θ is optimised numerically, e.g. via stochastic gradient descent). We discuss how to overfitting in more detail in Chapter 3. Underfitting, is mainly related to the way f_θ is specified. Two well known declinations of the general framework just mentioned that might be prone to underfitting are

i) the linear regression model, where we assumed $f_\theta(\cdot) := \langle \theta, \cdot \rangle$, with θ being in that case an unknown vector in $\mathbb{R}^D$ and the loss function usually taken as the mean squared error $L(y_i, \hat{y}_i) := ||y_i - \hat{y}_i||_2^2$;

ii) the logistic regression, which despite its name indeed is a *classifier*, with $f_\theta(\cdot)$ now being

$$f_\theta(\cdot) = \sigma\left(\langle \theta, \cdot \rangle\right), \tag{1.2}$$

where θ still is a vector in $\mathbb{R}^D$ and $\sigma(\cdot)$ denotes here the sigmoid **activation**

$$\sigma(x) := \frac{1}{1 + e^{-x}}.$$

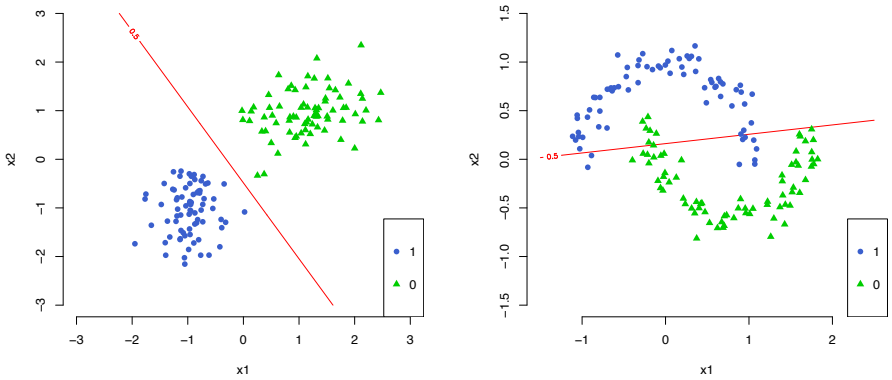
Since the logistic regression outputs predicted probabilities of belonging to one or the other class (i.e. $y_i \in \{0, 1\}$, for all i) it is customary to adopt the notation $\hat{y}_i := \hat{p}_i = f_\theta(x_i)$ and the the so called binary cross-entropy loss

$$L(y_i, \hat{p}_i) = -y_i \log\left(\frac{\hat{p}_i}{1 - \hat{p}_i}\right) - \log(1 - \hat{p}_i)$$

which is nothing but the negative log-likelihood of the Bernoulli distribution.

Both the above models are the standard **linear** tools for regression and supervised classification, respectively. Now, although linearity for the linear regression model is immediate to see, due to the presence of the sigmoid activation it might be less trivial to assess whether logistic regression is a linear classifier or not. Let us check why it is in a simplified example. Assuming that $D = 3$ and expressing Eq. (1.2) using log-odds, one has

$$\log\left(\frac{\hat{p}_i}{1 - \hat{p}_i}\right) = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2},$$



(a) Linear separability (b) Non linear separability

Figure 1.1: Two binary classification problems attacked with logistic regression (LR). The red line corresponds to the decision boundary obtained *after* fitting LR to the data. The data set on the left hand side is clearly linearly separable, whereas the one on the right is not. In that case, LR suffers from underfitting.

where $\theta := (\theta_0, \theta_1, \theta_2)^T$ and $x_i = (1, x_{i1}, x_{i2})^T$. Now, usually the data point x_i will be assigned to class 1 if $\hat{p}_i > 0.5$ or 0 if $\hat{p}_i \leq 0.5$. So the **decision boundary** is nothing but the set of points having a $\hat{p}_i = 0.5$. For such a generic data point x, via the above equation one has

$$0 = \log 1 = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

which is the implicit equation of a straight line, see Figure 1.1.

1.2 Multilayer perceptron

Linearity has some advantages (e.g. simplicity, explainability, sometimes closed formulas) but might be two simplistic and so lead to underfitting (Figure 1.1b). One way to go beyond the linearity assumption is to adopt **neural networks** in order to parametrise f_θ. The simplest (not trivial) neural net one might think of is the multilayer perceptron (**MLP**)

$$f_\theta(x_i) := \sigma_2 \left(W_2 \sigma_1 (W_1 x_i + b_1) + b_2 \right), \tag{1.3}$$

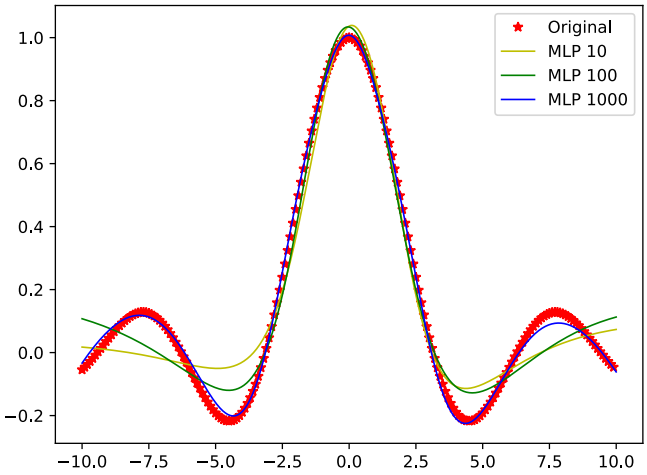


Figure 1.2: Approximation of the function $f(x) = \frac{\sin(x)}{x}$ via a MLP with one hidden layer with variable number of nodes $P = 10, 100, 1000$.

where $W_1 \in \mathbb{R}^{P \times D}$, $b_1 \in \mathbb{R}^P$ with P denoting denoting here the number of *neurons* or hidden layer size and $W_2 \in \mathbb{R}^{P \times K}$, $b_2 \in \mathbb{R}^K$. So the learnable parameters are $\theta := \{W_1, W_2, b_1, b_2\}$ and σ_i denote here *non*-linear activation functions (Section 1.3) which act element-wise. Here, K denotes the output size: in case of linear regression or binary classification $K = 1$ and $\sigma_2 = Id$. For multiclass classification, $K > 1$ and σ_2 is a softmax function.

Thanks to an additional notation we can reformulate the definition of f_θ recursively:

$$h_i^{(1)} := \sigma_1 \left(W_1 x_i + b_1 \right)$$
$$f_\theta(x_i) := \sigma_2 \left(W_2 h_i^{(1)} + b_2 \right).$$

This definition puts in light two important things: first an MLP is noting but a composition of linear layers interleaved by a non-linear activation function. Second, the above MLP has **two** layers of learnable parameters, namely (b_1, W_1) mapping the input (x_i) to the intermediate or hidden units $(h_i^{(1)})$ and (b_2, W_2) leading to the the output units $(f_\theta(x_i))$. More generally we can

define neural nets with an arbitrary number of hidden layers L

$$\begin{aligned} h_i^{(1)} &:= \sigma_1(W_1 x_i + b_1) \\ h_i^{(2)} &:= \sigma_2\left(W_2 h_i^{(1)} + b_2\right) \\ \cdots &:= \ldots \\ f_\theta(x_i) &:= \sigma_L\left(W_L h_i^{(L-1)} + b_L\right). \end{aligned} \tag{1.4}$$

Even without needing more than one hidden layer, MLP equipped with any non polynomial activation function can approximate any continuous function between two Euclidean spaces to any arbitrary precision, in the limit $P \to \infty$ (Universal Approximation Theorem[1]). Being a formal statement and proof of this claim behind the scope of this course, we play with it empirically in Figure 1.2. In red (stars) one sees the function $\frac{\sin(x)}{x}$ evaluated on a regular grid between -10 and 10. Three MLPs with one hidden layer were trained in order to approximate that function (MSE loss) with number of neurons in the hidden layer (P) varying from 10 to 1000. More details will be discussed in a dedicated Python notebook.

In order to conclude this short section on supervised learning and feed forward neural nets, we recall that, after plugging Eq. (1.3) into the minimisation problem (1.1), one has to solve a non convex optimisation problem which is usually attacked by means of stochastic gradient descent (Bottou, 2010), where the gradient w.r.t. θ can be computed via automatic differentiation (Baydin et al., 2018). More details are discussed in Chapter 2

Some vocabulary. Although in the deep learning literature the expressions "feed-forward", "fully connected" or "multilayer" are used interchangeably they are not synonyms. Indeed, feed-forward neural nets are all those networks whose architecture does not exhibit directed cycles (i.e. it is a DAG). Hence, residual or skipped connections, for instance, fall in this category. Fully connected neural networks are the ones we explored in this section, they might have two or more layers and no connection is skipped. Finally, a multilayer neural network is every neural net having at least two layers. With this definition, linear and logistic regression are examples of single layer neural networks.

[1] https://en.wikipedia.org/wiki/Universal_approximation_theorem

1.3 Activation functions

As we saw, non-linear activations are a key ingredient in the universal approximation theorem. In principle, one can adopt any non linear function as activation as long as it is i) non polynomial and ii) **differentiable** since, as we will see the optimization of θ relies on gradient based methods. We already encountered the **sigmoid** activation

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

allowing us to "convert" the final pre-activation unit in the MLP into a probability (binary classification). Although it can also be used to activate the units in the intermediate layers, this choice is no longer very popular since the sigmoid activation is more prone to **gradient vanishing** issues than other techniques.

Exercise 1.1 Show that $\lim_{x \to \pm\infty} \frac{d\sigma}{dx}(x) = 0$.

Another activation, strictly linked with the sigmoid is the **hyperbolic tangent**:

$$\tanh(x) := \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

Execise 1.2 Express the hyperbolic tangent as a function of the sigmoid $\sigma(x)$.

Exercise 1.3 (from C. M. Bishop and H. Bishop, 2023) Use the solution of the previous exercise to show that, given a two layer MLP as the one in Eq. (1.3), with a sigmoid activation after the first hidden layer, there exists an equivalent network computing the very same output but with hidden-unit activation given by $\tanh(\cdot)$.

Maybe the most used activation function for the hidden units is the so-called **ReLU**

$$\mathrm{ReLU}(x) := \max\{x, 0\}$$

which however is not differentiable in zero. We'll see in a Python notebook how the developers of PyTorch solved this issue. Another option, related with

ReLU is **softplus**

$$\text{softplus}(x) = \log(1 + \exp(x)),$$

a fully differentiable activation strictly related with ReLu.

Exercise 1.4 Consider the activation $\frac{1}{k}\text{softplus}(kx)$ and show that it converges to $\text{ReLU}(x)$ in the limit of k going to $+\infty$. Plot that activation against the ReLU for increasing values of k.

A third alternative is the **ELU** activation

$$\text{ELU}(x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha(\exp(x) - 1) & \text{if} \quad x \leq 0 \end{cases}$$

where α is a positive parameters fixed by the user.

Exercise 1.5 Show that ELU is everywhere differentiable iff $\alpha = 1$.

The final activation we mention is a special one since it is only used to activate the output layer of neural networks when performing multi-class classification. In this case each observation is assigned (exclusively) to one of $K > 2$ classes, so we want the network to output K probabilities summing to one. In general, given an input vector $x = (x_1, \ldots, x_K) \in \mathbb{R}^K$, it can be converted into a probability vector via the **softmax** activation

$$\text{softmax}(x) = \left(\frac{\exp(x_1)}{\sum_{j=1}^K \exp(x_j)}, \ldots, \frac{\exp(x_K)}{\sum_{j=1}^K \exp(x_j)} \right) \tag{1.5}$$

1.4 Loss functions

There are three loss functions mainly used in supervised learning, each corresponding to a specific task. For regression, the standard loss function is the mean squared error (**MSE**)

$$\mathcal{L}_N(\theta) := \frac{1}{N} \sum_{i=1}^N ||y_i - f_\theta(x_i)||_2^2,$$

where $f_\theta(x_i)$ is the neural net output layer for the i-th observations and y_i is a real vector in dimension K (possibly equal to one). Now, observing that $1/N$

is a constant in the above loss we easily argue that such a loss is equivalent to the negative log-likelihood of the following generative model

$$y_i \sim \mathcal{N}(f_\theta(x_i), I_K).$$

with $y_1, \dots, y_N$ being independent random vectors. Since the mean of Gaussian distribution here lives in $\mathbb{R}^K$, the output activation in the neural net will be the identity.

Exercise 1.6. Assume that $y_i \overset{\text{indep}}{\sim} \mathcal{N}(f_\theta(x_i), \Sigma)$ with Σ being a square covariance matrix of order K. Write down the corresponding loss function.

We already encountered binary cross entropy (**BCE**)

$$\mathcal{L}_N(\theta) := -\sum_{i=1}^{N} \left(y_i \log f_\theta(x_i) + (1 - y_i) \log(1 - f_\theta(x_i))\right)$$

which is the standard loss function in binary classification. Also in this case, the loss is nothing but the negative log-likelihood of the generative model $y_i \sim \mathcal{B}(f_\theta(x_i))$, independently, with $\mathcal{B}(\cdot)$ denoting here the Bernoulli distribution. Since the neural net $f_\theta(\cdot)$ now models a probability, the output activation is a sigmoid.

Finally, for multiclass classification problems, **categorical cross-entropy** is adopted

$$\mathcal{L}_N(\theta) := -\sum_{i=1}^{N} y_i \log f_\theta(x_i),$$

corresponding to the negative log-likelihood of independent categorical distributions $y_i \sim \text{Cat}(\mathbf{p}_i := (p_{i1}, \dots, p_{iK}))$, with $\mathbf{p}_i = f_\theta(x_i)$ and K denoting here the number of (mutually exclusive) classes. Since $\mathbf{p}_i$ lives in the $(K-1)$-simplex, the neural net $f_\theta(\cdot)$ is equipped with a softmax output activation.

As it can be seen, the general pipeline in supervised learning in as follows: the observations $y_1, \dots, y_N$ are assumed to be independent outcomes from a given distribution $q(\cdot)$ (Gaussian, Bernoulli, Categorical, etc...) depending on a parameter η_i specific to each observation (e.g. the mean of a Gaussian distribution or the probability of success of a Bernoulli), such that $y_i \sim q(\eta_i)$. Then, such a parameter is modelled via a neural net taking the corresponding feature as input, namely $\eta_i = f_\theta(x_i)$.

Exercise 1.7. (*Poisson regression*) Consider the case where $y_1, \ldots, y_N$ are positive integers and assume $y_i \sim \mathcal{P}(\lambda_i)$, independently, where $\mathcal{P}(\cdot)$ denotes here the Poisson distribution. Which output activation function would you choose for $f_\theta(\cdot)$? Write down the loss function.